Communications of the Association for Information Systems

Volume 22 Article 4

1-2008

Building Enterprise Architecture Agility and Sustenance with SOA

Minglun Ren
Hefei University of Technology, P.R. China

Kalle J. Lyytinen
Case Western Reserve University, kalle@po.cwru.edu

Communications of the Association for Information Systems

CAIS

Building Enterprise Architecture Agility and Sustenance with SOA

Minglun Ren

School of Management,

Hefei University of Technology

Hefei 230009,P.R.China

renml@mail.hf.ah.cn

Kalle Lyytinen

Department of Information Systems

Weatherhead School of Management

Case Western Reserve University,

Cleveland, OH 44106, USA.

Abstract:

Service-Oriented Architecture(SOA) is primarily regarded as a technical architecture consisting of tools and service specification to build loosely coupled applications. At another level it is also a means to leverage flexibility and agility to system services as it offers a hierarchical framework to coordinate simultaneous business process design and implementations using loosely coupled service infrastructures. SOA has been debated both in the academy and industry and misinterpretations of its nature impede its adoption. We summarize its historical origins and current evolutions. We review technologies that underlie SOA. In particular, we address how to integrate SOA initiatives with current technology platforms, and how to enforce reuse during the design of loosely coupled systems. We also analyze SOA design methodologies and platforms, and what are their roles in the application integration. Finally we outline challenges and future research directions for SOA.

Key words: SOA, Service-Oriented Architecture, loosely coupled system, reuse, business agility

Volume 22, Article 4, pp.75-86, January 2008

I. INTRODUCTION

Information system architectures have steadily evolved to be more adaptable to changing business environments. Scalability, agility and reusability have formed the mainstream of information system development improvements for last two decades. At the same time Information Technology (IT) has become strongly intertwined with business and it forms a critical part of business architectures that need to comply with the enterprise strategy. Currently, SOA (Service-Oriented Architecture) is deemed to offer a better way of specifying distributed computing services, identifying them and organizing them into loosely coupled applications. It is also expected to provide an effective means to address business agility, and it is being widely promoted for this reason. Yet, a majority of enterprises still consider that the adoption of SOA is difficult. There prevail also significant differences in understanding what SOA is and what can be done with it. Researchers, standardization organizations such as OpenGroup, or OASIS, and leading vendors like IBM all have come up diverse definitions of what SOA means to add to the confusion. Examples of definitions that abound are:

1. *A service-oriented architecture is essentially a collection of computational services*. These services communicate with each other and involve either simple data passing or two or more services coordinating some activity. Therefore generic means of connecting services are needed (see www.service-architecture.com).
2. *SOA is a style of IT architecture that delivers enterprise agility and boundaryless information flow.*(see www.opengroup.org/projects/soa/).
3. *Service-Oriented Architecture is a business-centric IT architectural approach that supports integrating business as a linked, repeatable business task, or service [IBM ,2003]*.
4. *Service-Oriented Architecture is a system for linking resources on demand*. In an SOA, resources are made available to other participants in the network as independent services that are accessed in a standardized way. This allows for flexible loose coupling of resources than is possible in traditional architectures (see www.looselycoupled.com).
5. *SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains*. It provides a uniform means to offer, discover, interact with and use computing capabilities to produce desired effects consistent with measurable preconditions and expectations [OASIS 2005].

While comparing these definitions, we can observe that definitions 1 and 2 treat SOA only at the technical level, while definitions 3 and 4 define SOA from a managerial perspective. Definition 1 emphasizes SOA services as a process of computational events, while definition 2 address SOA as an explicit and separately defined IT architecture. Definitions 3 and 4 view SOA in business-oriented and resource-based terms, respectively. Definition 5 —by OASIS—focuses on the concept of “needs and capabilities,” where SOA is seen to provide a mechanism that matches needs of users with the capabilities provided by service providers.

It is not unusual that there are many interpretations of any new computing “fad.” Different interpretations demonstrate that there are multiple aspects to consider when one seeks to make sense of an emerging complex technological innovation like SOA. At the same time, if we explore SOA more deeply and trace the idea back to its origins, differences between the definitions become justified. In short, SOA can be seen as a successor of multiple innovative ideas in computing, which it tries to integrate. Yet, to make SOA a sound basis for enterprise-wide IT strategy, a thorough assessment of its related concepts, technical basis, application methods, as well as its business potential is needed.

This paper seeks to address this need. We will review the evolution of the SOA concept and standards, and discuss its essential features. We also discuss how it can be applied and what are its future business and technological challenges. Accordingly, the remainder is organized as follows: Section II discusses the origins of SOA from both technical and managerial perspective. Section III reviews the state of the art in SOA designing and using, and addresses two key principles in SOA—reuse and loose coupling, respectively. Section IV summarizes principles of “service,” which is the most essential notion in SOA, and describes methodologies for SOA design and implementation. Section V analyzes platforms that support SOA delivery; while Section VI lays down challenges in SOA applications, and presents some suggestions for future research. Section VII offers conclusion remarks.

II. SERVICE ORIENTED ARCHITECTURES: ORIGIN AND EVOLUTION

SOA has been discussed in recent years in terms of technical baseline, enterprise architecture, implementation methods, and governance. These studies cover diverse areas such as SOA-based solutions for enterprise computing [Keen 2004; Mattern and Wood 2006], technical integration protocols for diverse applications [Erl 2004], services coordination [Leymann et al. 2002], as well as the need for new software infrastructures that integrate distributed systems [Krafzig et al. 2004]. Inroads have also been made in SOA's capabilities, its computational mechanisms and implementation strategies [Barry 2003]. Most of these researches view SOA within the boundaries of a single corporation. In addition, SOA analyses have not been extended to address enterprise heterogeneous interfirm platform integration and agile interbusiness reorganization. This primarily technical focus method makes SOA as currently conceived as a "flat" platform: It consists of application layer protocols and tools that will operate within a single organization for application integration. We will next review SOA from this narrower technical perspective.

Technical Perspective

SOA's core principles: scalability, agility, and reusability, have been the focus of IT development for some time. Several software architectures before SOA share similar principles including the client/server architecture, object-oriented development, multi-agent systems, and CORBA or DCOM component architectures. Table 1 illustrates main features of these technologies that preceded and shaped SOA. The impact of multiple software architectures partly explains also why there are so many interpretations of SOA. SOA forms a logical continuation in the evolution of architectural practices for application development to the extent that some have suggested that SOA is just a revival of ideas inherent in modular programming, event-oriented, and component-based design. Each of these technologies has contributed to dealing with software complexity at the level of strategic [Vitharana et al. 2004], architecture [Jain and Reddy 2004; Jain et al. 2001], and components design [Jain et al. 2006; Zdun et al. 2007], and made system architectures more agile and sustainable.

Table 1. Features Emphasized in Different System Architectures

Architecture	Reusability	Agility	Scalability
Object-oriented method	×		
Client/Server architecture		×	×
Multi-agent System	×	×	
DCOM/CORBA	×	×	×

Yet, we believe that a significant qualitative change took place when Web services and grid computing emerged and made computing services ubiquitous within a global and universal platform. Web service enabled SOA has solicited endeavors from various academic area such as computer science, information systems, and application domains. In the global Internet-based platforms, SOA has been the main driver in achieving system level application interoperability and cross-platform integrity. SOA helps thus build systems that are platform independent but can use services running on different platforms located anywhere. It thus can more effectively protect past investments in different system platforms. In line with this significant progress has been made on the selection and management of services [Jain et al. 2004 ;Yu et al. 2007], services orchestration [Peltz 2003], modeling and designing methodologies of web services-oriented architecture [Ferguson and Stockton 2005], which all have advanced technical capabilities of SOA.

Business Oriented Evolution

Architectures have also evolved from a business perspective. Long before SOA was proposed, business-oriented methods and architectures have already been applied. Among them event-driven architecture is still pursued by Oracle [Tsui et al. 2005] and Amazon [DeCandia et al. 2007]. Amazon has also integrated some SOA implementations to its business architecture.

The efforts to build a system that align with business needs can be traced back to early '70s. Already, Couger and Knapp addressed challenges of requirements modeling and tried to align systems requirements along with the business goals [Couger and Knapp 1972]. Designers have also tried to develop tools and methods to support business processes. In 1980s, these efforts became a trend to build application generators, which related to specific business processes or functions [Xiao et al. 1993]. Unfortunately, these efforts did not meet all design goals because chosen system platforms were closed and too rigid. Though common data management platforms, especially relational databases enabled data integration, mapping business processes into agile software functions which would be decomposable and configurable remained a challenge. In the 1990s, efforts have been made to automate business process by generic workflow models [Edward and Zhao 2001].

However, workflow systems have often been too rigid and thus failed to address needs of organic and volatile processes common in many businesses [Abbott and Sarin 1994]. The ERP solutions offered one way to integrate organization-wide business processes based on a singular process framework offered by a singular ERP vendor. This increased scale and flexibility but did not offer ways to configure heterogeneous computing resources in an agile way. Business processes have also been researched using an activity-based approach which enhances some IT tools like e-mail with business semantics. These facilitate access to people, data and other resources, so as to support collaborative work [Moody et al. 2006]. The recent wave of web services and agent-based systems have offered a new way to address business change using flexible technical platforms.

Web services defined on a technical level can more easily fulfill varying process demands. Michael P. Papazoglou has therefore proposed an extended SOA architecture to leverage organizational agility [Papazoglou 2005]. IBM Research has likewise developed a model-driven architectural framework for transforming business processes to be service-oriented [Kumaran and Bhaskaran 2005]. So far, business semantics have been added afterward on top of the existing platforms to simplify modeling and to avoid hiding business semantics behind too many technology details. Accordingly, the idea of building business-layer modules that can be configured flexibly has become more desirable and has been promoted in new architectural approaches. Overall, the evolution of architectures has resulted in cross platform standards, flexible application communication protocols, and integrated suites of business analysis tools to support business change.

As a whole, SOA makes it possible to combine technical and business aspects while planning an application architecture using user defined services. These services are expressed in business related terms. By providing a business-oriented view SOA helps users interact with business level services and analysis. By mapping these business services to software services makes software development to become similar to modular process assembly. Users can deploy services from multiple providers when these services fit with their needs and existing platform standards.

III. STATE OF THE ART IN DESIGNING AND USING SOA

SOA's main principles were presented in SOAD (Service-Oriented Analysis and Design) guidelines [IBM 2006]. In this report, SOMA (Service-Oriented Modeling and Architecture) is outlined as way to analyze SOA services. Since then, commercial vendors have provided additional principles in building such architectures. For example, in August 2006, OASIS suggested a reference model of SOA (SOA-RM), which formally presents principles of service composition, description, contracts and execution, as well as run-time dynamics of services such as visibility and interconnection (see www.oasis-open.org). SOA-RM can be viewed currently as the summary of best practices in the service design area. It does not specify the exact implementation platform or detailed standards for service announcements, contracts and invocation, but defines a generic SOA framework to integrate applications. Because single uniform technical platforms are not the aim of SOA, it can be applied in the context of earlier architectures that have preceded available implementations of SOA. Ideas of services were already present in CORBA and DCOM as both of them use notions of services, contracts, and Remote Procedure Calls (RPC). These architectures were not, however, described from nor aligned with a business perspective.

Web services standards formulated by W3C have recently provided a complete set of communication standards for generic SOA implementations including XML (for data structuring), SOAP (for service calls), REST (for service calls), UDDI (for service registration), and WSDL (for service description). These standards will "drive SOA to the mainstream" [Gartner 2003] as the mechanisms they promote are suitable across the Internet. The recent ratification of WS-BPEL (Web Services Business Process Execution Language) by OASIS enables users to describe business process activities as Web services and specify how they can be connected to accomplish specific tasks (see www.oasis-open.org). This provides an additional catalyst for Web services design which is aligned with business processes and leads to a common way for practitioners and academic researchers to approach process design and as service design [Vitharana et al. 2007]. With more services being built at different pockets of the community, and as the service performance improves, the enterprise computing will move gradually to service-level orchestration and outsourcing. Web 2.0 movement has advanced some additional principles for Web services like integrating rich services and data from multiple sources and making them available to users and other applications [O'Reilly 2005]. Web 2.0 applications are regarded by many as the first wave of mature SOA solutions, though there are still questions to what extent these applications will comply with enterprises architecture.

Due to the proliferation of SOA platforms the Web service applications have become more common across many industries. Many of them use now Web services and peer-to-peer integration. This demonstrates that SOA can be used as a way to integrate heterogeneous computational processes across enterprise borders. Though these applications have brought increased business value, they have been mostly used for singular and narrow business cases without a strict compliance to enterprise architectures. In fact, very few implementations have taken place so far in the context of an overall enterprise wide SOA strategy [Wainewright 2003], though over 60 percent of enterprises since 2005 have begun their SOA projects, or are currently considering to adopt SOA as a means to integrate their applications and provide new services to their customers or partners [Baroudi and Halper 2006]. At the same time, nearly half of the organizations which have applied SOA remain dissatisfied. The reasons are multiple: the lack of planning and clear business case, lack of understanding of what services are available, the lack of governance, and the lack of standards. Moreover, building generic software services that are suitable for many situations cost more to build than building special purpose services and are thus difficult to justify financially. Though savings during reuse can reach 90 percent of the estimated delivery cost, the ROI of SOA is determined by the reuse frequency for each service over a set of services, which is difficult to anticipate [Poulin and Himler 2007]. Despite this some estimates suggest that around 2010 SOA will become a mainstream practice and will end the 40 years domination of monolithic software architectures [Gartner 2003]. This is largely based on optimistic interpretations of how reuse will be supported and enhanced in SOA through principles of service reusability and loose coupling.

Reuse of Services in SOA

SOA suggests that reuse of services will develop significantly, as reuse will be located onto the highest level—systematic reuse—in the hierarchy of reuse policies:

1. *Component reuse*. This kind of reuse emphasizes reuse at code level. It offers a wide range reuse methods, including component-based design, object-oriented methods, agent-based systems etc. [Brazier et al. 2002] A common trait among these technologies is that the reusable code is encapsulated into independent modules, and the reuse occurs through procedure calls, references to shared parameters, or by inheritance. All these make the reusable components substitutable, which can be assembled together.
2. *Context reuse*. This kind of reuse has been more successful and involves reuse technologies such as middleware, infrastructure, frameworks etc. These approaches put an emphasis on services and capabilities. The reuse is realized by applying multipurpose assets like architectures, patterns, components, and frameworks. The framework affects software organization and capabilities, and reduces development time and cost while improving software quality [Schmidt 1999; Biddle et al. 2003].
3. *Systematic reuse*. Systematic reuse is carried out at the *business level*, so that services across enterprise can be shared as to assemble business processes which adapt to business change. These reuse units are call *services,* which have a clear business meaning and explicit functional boundary. Moreover, these services can be loosely coupled, and therefore suitable for scaling up and handling change.

The reuse of services in service-oriented software has a natural extension to outsourcing of services. Computing and storage resource leases have now become standard due to associated economies of scale and scope. Google's offering of e-mail or office services is a case in point. For many enterprises challenges associated with the increased need for agility software service leasing will be the next natural step (software as service). With the development of grid computing services reuse will become a natural way to outsource and compose complex applications. Since services in SOA are loosely coupled and distributed, the following two aspects will be crucial in effective SOA application delivery: reusability and consumability.

Reusability is expected from most software services. The services cannot be implemented only to meet a narrow situation but also to address multiple service situations. One increasingly popular development approach is to code first what the user needs currently, but then when new extraneous functionality is needed to continually streamline and re-package the code until it is general enough to handle multiple business cases. Yet, the basic idea of service reusability does not guarantee that reuse can be automatically achieved. Service consumability must also go up. This is about mechanisms that enable application builders to properly recognize, identify, and access reusable services. To increase consumability, software developers need frameworks and associated search and integration tools by which services can be integrated across all stages of a system lifecycle. For example, WSDL specifications, service models, and tools related to service configurations and orchestration have become critical for promoting consumability.

As noted SOA relates closely to the need to support and implement flexibly enterprise level processes. Therefore, reusable services must make sense from a business perspective. Yet, a service meeting multiple business needs and a framework to connect and re-configure services are not enough. For effective deployment of SOA applications, a fundamental challenge is to determine an economic level of granularity for reuse: how fine or coarse should a service be, and how to combine atomic services into molecular services, and then integrate them into business models and processes. This should be done simultaneously in ways which does not lead to too rigid hierarchical systems. However, there is currently very little work on the economics of service encapsulation and implications of different principles of granularity for effective service composition and reuse.

Loosely Coupled Services

"Loosely coupled" is a systemic feature of SOA, which is assumed in almost all definitions of SOA. Critical properties of loosely coupled systems at the architecture level have been investigated for years. For example J2EE, .NET both played an important role in building alterative architectures for loosely coupled software systems. In all these only behaviors of services are specified in their external descriptions, whereas internal service implementations are treated as "black boxes." Thus, a "loosely coupled" service should be called remotely without critically affecting the overall architecture, state or behavior of the system. In addition, these behaviors are "loosely coupled" during the service orchestration while composing and running the application through middleware software brokers. Yet, what one means by this loose coupling in practice is not always clear. As specified in the SOA definition, a SOA application seeks to deploy distributed services across the whole Internet. In order to do so one needs three types of agents to operate effectively in specified roles (not just software components): 1) service providers; 2) application developers; and 3) service brokers. In addition, each of them operates in an uncertain and dynamic environment as each service can be reused by a number of customers in different ways. Likewise, application builders must have the capability of recognizing, contracting and interoperating services, even though all properties of these services are not known at the time of the application composition, which raises the issues of trust, responsibility and liability.

One purpose of "loose coupling" is to achieve flexibility: loosely coupled systems assume internal integrity of components and seek to apply these components in interoperable and flexible ways. This demands that business processes can be decomposed into separate (software) services, which can be then flexibly combined. In addition, SOA allows connect services across companies and service deployment is not confined to internal assets. The links that services allow are of many sorts: company-to-company, activity-to-activity, agent-to-agent, or person-to-person. Thus, each individual service can be plugged into the business process which makes the business process change more flexible, enabling new types of collaborations between businesses [Haugen 2000].

So far only a few loosely coupled systems as defined in the SOA standards have been built. This is partly because the increased complexity of functions and behaviors of service components has not been addressed adequately in current platforms, and partly because the traditional idea of reusable components has a different granularity and composition logic. SOA strives for loosely coupled business systems and thus involves a different level of granularity. Basically, it echoes the shift toward loosely coupled industrial organization, which needs to be supported by adequate and flexible IT capabilities [Sharp and Ryan 2005]. Therefore, a loosely coupled system in SOA has the following features:

1. *Business orientation*. SOA seeks to make services easier to deploy from a business perspective. When business functions are implemented as services, the linkages between functions can be verified by business logic, and by the information transferred between business activities. So business orientation is not only demanded but is regarded as the only feasible way to build loosely coupled systems.
2. *Various abstraction levels*. Services on the same abstraction level are more convenient to integrate. Universal communication methods can be used to achieve interoperability. Unfortunately, with heterogeneous processes and applications, services with different granularities exist. Therefore the goal should be to define each service within a clear application level discipline and have its interfaces defined only at a certain level. This demands that a hierarchical information model must be built of the whole enterprise—a goal which has remained elusive.
3. *Explicit boundary definition*. Clearly defined services should have explicit boundaries so that their function, data creation and utilities are limited and have a well defined scope. This makes service implementation easier whereby necessary service references can be more clearly described.
4. *Service autonomy*. In SOA, a loosely coupled system requires that distributed services are autonomous and seen to act as black boxes. They should not be affected by other services. This makes interactions between services totally dependent on references that are acquired through explicitly defined interfaces.

These four loose couplings illustrate applicable rules of service recognition and modeling in service-oriented systems. "Business orientation" describes a service from an application layer and specifies its objectives and functions. "Abstraction levels" demonstrate the hierarchical heterogeneity of services, which mirror multiple information processing levels in enterprises. "Explicit boundary definition" describes the component based feature of SOA and illustrates the separability of business processes and functions. "Service autonomy" describes business transactions as elements in business processes. Thus, these features allow viewing a business process as a multi-level information processing task made up of a series of independent services that have clear business semantics. Services fulfilling these requirements help orchestrating services in a SOA application.

IV. DESIGN PRINCIPLES OF SOA

Generic Design Goals

We will next describe the five goals of SOA application design which can be achieved by a disciplined application of the principles of systematic reuse and loose couplings [Vinoski 2002; Actional 2005]:

1. *Increased agility.* This is a main reason for enterprise managers to use SOA as they need to deal with fast business change. They hope to solve this just by investing in SOA capabilities.
2. *Heterogeneous and flexible outsourcing*. IT capabilities and services can be identified and sourced from multiple sources, and in flexible ways. This type of outsourcing will not only reduce the development cost, but will also reduce delay, and dramatically accelerate implementation.
3. *Protecting investments*. Many enterprises have invested in large legacy systems for decades. Therefore any decision to introduce a new system architecture should consider the reuse of current assets.
4. *Scaling up for building truly enterprise-wide applications*. The loosely coupling makes increasingly flexible design possible. Enterprises can choose to design and implement suitable and profitable processes and capabilities across the enterprise, while not needing to worry about the integrity of the whole suite of enterprise applications.
5. *Improving sustainability*. The rapid progress of IT shortens the life cycle of systems. SOA can make systems and their implementations more sustainable.

In the history of system design there is hardly anything new in these design aims, but SOA environments add new elements to them, which makes SOA promising in fulfilling these design aims. The only new feature of SOA seems to be its open-ended and loosely coupled nature and business-orientation. Yet, the latter is omitted in most discussions, which demonstrates that SOA is widely still viewed as a technical approach rather than a business integration mechanism. The design principles of SOA can be accordingly classified into three categories of system, service and business design features:

System features
1) Independence from technical platform
2) Loose coupling
3) Reusability
4) Interoperability

Service features:
5) Encapsulation
6) Autonomy
7) Deceivability
8) Contract

Business design features:
9) Services carry business semantics that can be understood by business people;
10) Services have the ability to be composed as a business process; and
11) Services are defined from the view point of enterprise integration.

SOA Design Methodologies

Methodologies play an important role in putting any design principles to enterprise-wide use. Though many uncertainties prevail in how to implement SOA principles, several applications of SOA have garnered guidelines of how to plan with and by SOA, how to extract services from business functions, and how to make SOA principles operational. For example, SOMA (Service-Oriented Modeling and Analysis) addresses SOA analysis and design by focusing on how to identify, specify, and realize services. SOMA adopts two approaches in analyzing the service architecture: One is top-down, in which the business processes are analyzed to make them compliant with the enterprise architecture. This approach analyzes business processes, and then decomposes processes into sub-processes and tasks and thus builds a business perspective of services and applications. The other is bottom-up approach, which seeks to find useful services to complement the top-down analysis and to wrap legacy system into services as to protect investments [Lawler and Howell-Barber 2007].

OASIS emphasizes decomposition of business areas and interactions between them. The OASIS SOA Adoption Blueprint defines a service decomposition approach and provides a notation to present services that is similar to UML Use cases. The OASIS approach starts at a functional level and examines key functional business areas (see www.oasis-open.org).

Amazon's event-driven method is an example of a design method that emerged from practical needs. Amazon's strategy is somewhat a hybrid. Software designers are responsible for the recognition of services that will improve customers' satisfaction. They must also design services and make them useful and available for others. Amazon has developed many services in this way and seems to have succeeded in composing services in systematic ways [Vogels 2005].

Another concern is whether to build SOA applications as a monolithic system and use all services within an single enterprise or whether to use SOA services to serve external customers and collaborators. This implies a decision: whether to develop all services by oneself, or whether to outsource some of them. Different goals affect also the platform selection: how interfaces and connections of services are described and implemented, and the overall implementation strategy. Some large vendors like IBM have emphasized outsourcing [IBM 2001]. In contrast, OASIS does not mention whether these services should be self-designed or outsourced, while Amazon limits their services to their own services [Vogels 2005]. If we consider the globalization and demands for the agility, it is obvious that outsourcing will become essential for system building in future. This will bring a new aspect to IS design and raise more interests to developing SOA methodologies that recognize design of public methods and services, how to present varying user interfaces, and how to embody interoperable features, and address platform independence during design time [Yourdon 2004].

V. SOA PLATFORMS

SOA is not bound to any specific platform. There is, however, a need for relatively complete and fixed service-oriented features in any platform, if it is expected to play an important role in the application of SOA. Organizations need frameworks implemented in platforms that embody SOA principles and which help integrate, identify and configure distributed services. A platform that has these functions will be called a SOA platform.

Web Service

Web service is the most popular technical architecture applied in SOA, among which the most popular and commercially successful platform is SOAP messaging protocol. In SOAP Web service invocations must be carried that are compliant to SOAP messaging standard. Services are described in a service description standard called WSDL. Since SOAP allows multiple message exchange patterns such as request-and-response, broadcasting and sophisticated message correlations, it can be used to integrate almost all kinds of legacy system from different vendors (see www.w3.org).

SOAP protocol is an example of Remote Procedure Call (RPC) paradigm. When a RPC message is coded into a SOAP message, it provides a tunnel for message transfer between services. This type of RPCs is specific, and can not be reused in other implementations. So SOAP alone is seldom viewed as a SOA framework.

Another set of Web services can be obtained through the application of REST. REST web services are light weight and require little infrastructural investments besides standard HTTP and XML technologies, which are now well supported by most programming platforms[He, 2003]. REST web services are simple and effective, because HTTP is the most widely available service interface protocol, and it is good enough for most applications. HTTP does not need introducing an additional transport layer like SOAP, and in many cases it provides simple transport of messages.

CORBA/DCOM

CORBA(Common Object Request Broker Architecture) forms a standard architecture for distributed object-oriented systems. It allows a distributed, heterogeneous organization of distributed objects to interoperate. In CORBA architecture, an Object Request Broker implements the requests to apply services of a remote object. It locates the remote object on the network, communicates the request to the object, waits for the results and, when available, communicates the results back to the client. IDL (Interface Definition Language) provides the specification of object interfaces, and makes ORB an open architecture for integrating software components. IDL can be used to wrap existing code as objects in ORB. It demonstrates two important features of scalability and agility. The distribution of objects makes it easy to reuse objects, provided that the objects are well defined for business activities [Vinoski 1997]. So CORBA is technically a SOA architecture but needs business oriented analysis for distributed object design.

DCOM (Distributed Component Object Model) enables software components to communicate and has been used by developers to create reusable software components. It takes advantages of Windows object services. It uses a similar technology as SOA. Recently, DCOM has been replaced by .NET technology. Microsoft also provides SOAP Web service on .NET and it is suitable as service oriented architecture (see www.oasis-open.org).

SOA Development Tools

Some analysis and software design tools based on UML standard such as Rational Rose and Sphere modeler are widely used in the analysis of processes [Channabasavaiah et al. 2003]. However, practitioners would rather use a tool that offers a modeling environment to represent communications between services, management of services, and scheduling of services. Thereby an application system can be developed more rapidly. One example is Vinci, an architecture for building Web applications [Agrawal et al. 2002]. Likewise, Nimrod/G is a tool for resource management and scheduling in global computation grid [Buyya et al. 2000]. IONA and Eclipse are now working on a SOA Tools Platform (STP), which aims at building frameworks and extensible tools that enable the design, configuration, assembly, deployment, monitoring, and management of software services (see http://www.eclipse.org/stp). With the increased use of SOA in enterprise wide system projects, we can expect that vendors will provide advanced tools for rapid construction of applications and services for specific industries, processes or capabilities.

VI. CHALLENGES AND FUTURE RESEARCHES

SOA is not only a new technology to build information systems, but also a new computing architecture that can help to organize and integrate distributed computing resources. Since SOA is built on the open access in Internet, loose coupling, reuse and relies on distributed services, it will inevitably lead to new families of applications. The future systems compliant to SOA architecture will have new additional features as: (1) resources are organized and used based on social roles and responsibilities; (2) services will be the basic reuse unit; (3) services are distributed across the Internet; and, (4) Services are organized closely related to business logic. Despite steady advances in satisfying these requirements there remain several challenges confronting a wide scale SOA application. This demonstrates the current relatively immature level of its evolution. The following problems are observed widely in current research in SOA:

1. *Mechanisms for deploying reuse and loose coupling of services within regional or global area*. These include mechanisms for finding, sharing, negotiating, and interacting among services providers and system builders etc. Specific problems related to SOA include its security, interoperability and dynamic manipulation of data, etc. For example, UDDI has been extended in various Web Service applications [McIlraith and Martin 2003]. Efforts have been made to ratify Web services standards such as UDDI v2 and BPEL into OASIS standards, which will become helpful in service publication and business process description. ebXML is also used to share business-to-business information and security policies (see www.oasis-open.org). However, there are multiple platforms and tools to build SOA systems. Thus, more general standards and mechanisms for coordinating services from all kinds of platforms are needed and uniform description standards should be agreed and enforced to make SOA truly applicable across platforms.
2. *New design methodologies for loosely coupled systems*. Though information systems change over time from bottom up, most systems assume hierarchical designs. No theoretical and effective decompositions of processes according to the nature of information flow is currently available. Current research has offered some important results on design principles for service and operation designs [David and Artus 2006], but they still use similar methodologies as OOD or CBD, which are not closely related to business semantics. Global distributed service reuse can be viewed as a new type of business between partners. Accordingly, services supporting different business tasks should be built with different granularities. It is essential to establish a commonly accepted specifications of business levels and granularities, so as to specify what levels in a business system can and need to be designed as services [Bumer 2007].
3. *SOA platforms and design tools.* Building service-oriented architecture involves the analysis of enterprise architectures, the overall architecture of SOA, business process recognition, service recognition, service design and implementation, and maintenance. Current platforms focus on technical aspects, and leave a lot for application builders. For example, how to transform business functions into services? Current research is trying to more closely link SOA design with business development as supported by platforms such as IBM's WebSphere [Lv et al. 2007; Georgiev et al. 2007]. These would help build new features suitable for business-oriented services into SOA platforms. The other direction is to provide modeling tools that can decompose processes into services, help evaluate the design results and simulate process redesigns. These would help simulate service interoperations, manipulation of data, interfaces between services and evaluate performance and non-functional features like security and reliability [Afshar et al. 2005].
4. *Economic and organizational challenges in deploying SOA*. Understanding the economics of the new services is crucial. To meet the strategic needs of fast changing enterprises, costs and benefits of using distributed services should be carefully considered as a motivation to build SOA systems. Organizational changes will involve an implementation risk in adopting SOA. As services become distributed and are possessed and monitored by autonomous and distributed owners, reliability, trust and liability and constant performance evaluation will become more critical and complex when compared with traditional monolithic systems [William 2003].

5. *Domain ontologies aligned with service-oriented process design.* Generally, information systems are domain specific and need domain specific standards to facilitate service planning, interface design, service description and publication. Thus, business domain ontologies are crucial for the interoperability and wide scale adoption of SOA [Vitharana et al. 2007]. Though many studies have been carried out about domain ontologies and progress has been made in some areas like healthcare [Bell et al. 2007], a lot remains to be done. The standardization of domain specifications will lend SOA both with autonomy and compatibility. This will be the trend for adopting SOA in different industries, and, if successful, it will eventually make SOA a means to orchestrate different services complying with industry level service and ontology standards. Future research should therefore examine the increased agility and performance permitted by standardized architectures using shared domain ontologies.

VII. CONCLUSION

Current applications of SOA are far more advanced than current research and theory in their respective fields. As SOA applications have been developed in multiple industries, they are not only limited by software platforms, but also lack of a sound theoretical basis. As SOA was formulated as a general paradigm to orchestrate computational distributed services, many new application patterns are likely to emerge as to fully deploy the potential of SOA. To meet this goal breakthroughs in business process and design theory and orchestration, generic SOA design methods and tools, as well as standardization and implementation strategies to propel economic widespread use of SOA application are sorely needed.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of NSFC (Grant No.70631003).

REFERENCES

Abbott, K. and S. Sarin. (1994). “Experiences with Workflow Management: Issues for the Next Generation,” Proceedings of the ACM Conference on Computer supported Cooperative work (CSCW’94), ACM Press, New York,1994, pp.113-120.

Actional. (2005). White paper: “Implementing a Successful Service-Oriented Architecture Pilot Program,” www.actional.com/resources/webinars/SOA-Pilot.

Afshar, M., B. Nainani, and J. Raj. (2005). “Supporting the Business Process Lifecycle Using Standards-Based Tools,” *SOA World Magazine*, http://soa.sys-con.com/author/5212afshar.htm.

Agrawal, R., R. J. Bayardo Jr., D. Gruhl, and S. Papadimitriou. (2002). “Vinci: A Service-Oriented Architecture for Rapid Development of Web Application,” *Computer Networks* 39(2002), PP.523-539.

Baroudi, C. and Dr. F. Halper. (2006). “Executive Survey: SOA Implementation Satisfaction,” November 27, 2006, www.mindreef.com/report.

Barry, D. K. (2003). *Web Services and Service-Oriented Architectures: The Savvy Manager's Guide*, Morgan Kaufmann Publishers.

Bell, D., S. de Cesare, N. Iacovelli, M. Lycett, and A. Merico. (2007). “A Framework for Deriving Semantic Web Services,” *Information Systems Frontiers* 1(9),pp.69-84.

Biddle, R., A. Martin, and J. Noble. (2003). “No Name: Just Notes on Software Reuse,” *ACM SIGPLAN Notices* 38(12), pp.76-96.

Brazier, F., C. M. Jonker, and J. Treur. (2002). “Principles of Component-Based Design of Intelligent Agents,” *Data & Knowledge Engineering* 1(41), pp.1-27.

Burner, M. (2007). “Service Orientation and Its Role in Your Connected Systems Strategy,” http://msdn2.microsoft.com/en-us/library/ms954826.aspx#srorientwp_topic1.

Buyya, R., D. Abramson, and J. Giddy. (2000). “Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid,” Proceedings of The Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, vol.1, pp.283-289.

Channabasavaiah, K., K. Holley and E. Tuggle, Jr. (2003). “Migrating to a Service-Oriented Architecture,” *IBM DeveloperWorks,* http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design/.

Couger, J. D. and R. W. Knapp. (1972). *System Analysis Techniques*, John Wiley & Sons.

David, J. and N. Artus. (2006). “SOA Realization: Service Design Principles,” *IBM developWorks*, http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design/.

DeCandia, G. et al. (2007). "Dynamo: Amazon's Highly Available Key-Value Store," Proceedings of 21 ACM SIGOPS Symposium on Operating Systems Principles, Stevenson, Washington, USA pp.205-220.

Edward A. S. and J. L. Zhao. (2001). "Workflow Automation: Overview and Research Issues," *Information Systems Frontiers*,3(3), pp.281-296.

Erl, T. (2004). *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*, Prentice Hall.

Ferguson, D. and M. Stockton. (2005). "Service-Oriented Architecture: Programming Model and Product Architecture," *IBM Systems Journal* 2005, 4(44), pp.753-780.

Gartner. (2003). *Java Strategies & Technologies Research Note T-19-6412* M. Driver.

Georgiev, L., J. Ovtcharova, and I. Georgiev. (2007). "Modelling Web Services for PLM Distributed System,"*International Journal of Product Lifecycle Management* 1(2),pp.30-49.

Haugen, R. and W. E. Mccarthy. (2000). "REA: A Semantic Model for Internet Supply Chain Collaboration," Proceedings of the Business Objects and Component Design and Implementation Workshop VI: Enterprise Application Integration.

He, H. (2003). "What Is Service-Oriented Architecture?" http://www.xml.com/pub/a/ws/2003/09/30/soa.html.

IBM. (2001). "Outsourcing Services: A Powerful Strategic Tool for E-Business," http://www-05.ibm.com/services/be/pdf/so.pdf.

IBM. (2005). "Service Oriented Architecture," http://www-306.ibm.com/software/solutions/soa/.

IBM. (2006). "Elements of Service-Oriented Analysis and Design—An Interdisciplinary Modeling Approach for SOA Projects," http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/.

Jain, H. and B. Reddy. (2004). "Layered Architecture for Assembling Business Applications from Distributed Components," *Journal of Systems Science and Systems Engineering (JSSSE)* 1(13), pp.257-278.

Jain, H. et al. (2001). "Architecture for Deployment of Distributed Business Components to Assemble E-Commerce Applications," Proceedings of Sixth INFORMS Conference on Information Systems & Technology, Miami Beach, Florida.

Jain, H., H. Zhao, and N. Chinta. (2004). "A Spanning Tree Based Approach to Identifying Web Services," *International Journal of Web Services Research* 1(1), pp.1-20.

Jain, H., K. Rothenberger, and M. Sugumaran. (2006). "Flexible Software Component Design Using a Product Platform Approach," *Proceeding of International Conference on Information Systems (ICIS06)* Milwaukee.

Keen, M. (2004). "Patterns: Implementing an SOA Using an Enterprise Service Bus," *IBM*.

Krafzig, D., K. Banke, and D. Slama. (2004). *Enterprise SOA: Services-Oriented Architecture Best Practices*, Prentice Hall.

Kumaran, S. and K. Bhaskaran. (2005). *Business Process Integration. in: Supply Chain Management on Demand: Strategies,Technologies, Applications*, Chae An and Hansjorg Fromm (eds.), Springer-Verlag.

Lawler,J. P. and H. Howell-Barber. (2007). *Service-Oriented Architecture: SOA Strategy, Methodology, and Technology*, Auerbach Publications, New York.

Leymann, F., D. Roller, and M. T. Schmidt. (2002). "Web Services and Business Process Management,", *IBM Systems Journal* 2(41),pp.198-226.

Lv, W., J. Kang, W. Chen, and R. Lei. (2007). "Integration and Application Platform of Service-Oriented Telecom Businesses," *Fourth European Conference on Universal Multiservice Networks (ECUMN'07)*, pp.183-189.

Mattern, T. and D. Wood. (2006). *Enterprise SOA: Designing IT for Business Innovation*, O'Reilly.

McIlraith, S. A. and D. L. Martin. (2003. "Bringing Semantics to Web Services," *IEEE Intelligent Systems* 1(18),pp.90-93.

Moody, P. et al. (2006). "Business Activity Patterns: A New Model for Collaborative Business Applications," *IBM Systems Journal* 4(45), pp.683-694.

OASIS. (2005). "SOA Reference Model /TC," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

O'Reilly, T. (2005). "What Is Web 2.0? Design Patterns and Business Models for the Next Generation of Software," http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

Papazoglou, M. P. (2005). “Extending the Service-Oriented Architecture,” *Business Integration Journal* February, 2005, pp.17-21.

Peltz, C. (2003). “Web Services Orchestration and Choreography,” *Computer,* 10(36), pp.46-52.

Poulin, J. and A. Himler. (2007). “The ROI of SOA Relative to Traditional Component Reuse,” www.logiclibrary.com/pdf/wp/ROI_of_SOA.pdf.

Schmidt, D. C. (1999). “Why Software Reuse Has Failed and How to Make It Work for You,” *The C++ Report*.

Sharp, J. H. and S. D. Ryan. (2005). “A Review of Component-Based Software Development,” *Proceedings of the 26th International Conference on Information Systems*, pp.151-161.

Tsui, F. et al. (2005). “Key Design Elements of a Data Utility for National Bio-surveillance: Event-Driven Architecture, Caching, and Web Service Model,” *Proceedings of the 2005 AMIA Annual Symposium,* Washington, DC, pp.739-743.

Vinoski, S. (1997). “CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments,” *IEEE Communications Magazine* 2(35),pp.46-55.

Vinoski, S. (2002). ”Middleware ’Dark Matter,’” *IEEE Internet Computing* 5(6), pp.92- 95.

Vitharana, P., H. Jain, and F. “Mariam” Zahedi. (2004). “Strategy-Based Design of Reusable Business Components,” *IEEE Transactions on Systems, Man and Cybernetics—Part C Applications and Reviews* 4(34), pp.460-475.

Vitharana, P., K. Bhaskaran, H. Jain, H. J. Wang and J. L. Zhao. (2007). “Service-Oriented Enterprises and Architectures: State of the Art and Research Opportunities,” *13th Americas Conference on Information Systems(AMCIS2007,* Keystone, Colorado,2007, pp.1-11,

Vogels, W. (2005). “Amazon.com: E-Commerce at Interplanetary Scale,” *O'Reilly Emerging Technology Conference 2005*, http://conferences.oreillynet.com.Wainewright, P. (2003). “Tactics, Not Strategy, Drive SOA Adoption,”, http://www.looselycoupled.com/stories/2003/tactics-soa-infr0415.html.

William A. E. (2003). “An Evaluation Framework for Deploying Web Services in the Next Generation Manufacturing Enterprise,” *Robotics and Computer-Integrated Manufacturing* 6(9),pp.509-519.

Xiao, J., Z. Wang and L. Cao. (1993). “An I-CASE System for Automatic Generation of MIS Software—MISG/CASE,” *Proceedings of 1993 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering*, Beijing,1993, pp.407-410.

Yourdon, E. (2004). *Outsource: Competing in the Global Productivity Race,* Prentice Hall.

Yu, T., Y. Zhang and K. Lin. (2007). “Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints,” *ACM Transactions on the Web*, 1(1), Article 6.

Zdun, U., C. Hentrich, and S. Dustdar. (2007). “Modeling Process-Driven and Service-Oriented Architectures Using Patterns and Pattern Primitives,” *ACM Transactions on the Web* 3(1), pp.14-43.

ABOUT THE AUTHORS

Minglun Ren is associate professor of Information Systems at Hefei University of Technology, P.R. China. His work focuses on the use of information technology to support business innovation, decision making, as well as regional information.

Kalle Lyytinen is professor of Information systems at the Weatherhead school of management at Case Western Reserve University and an adjunct professor at University of Jyvaskila in Finland. He has published more than 150 academic papers and eight books. He is well known for his research in computer-supported system design and modeling, systems failures and risk assessment, computer supported cooperative work, and the diffusion of complex technologies. He is currently researching the development and management of digital services and the evolution of virtual communities. Kalle is now the editor-in-chief of the *Journal of AIS*.

Communications of the Association for Information Systems

ISSN: 1529-3181